Improving Recurrent Neural Networks with Predictive Propagation for Sequence Labelling

Son N. Tran[1(✉)], Qing Zhang[1], Anthony Nguyen[1], Xuan-Son Vu[2], and Son Ngo[3]

[1] The Australian E-Health Research Centre, CSIRO, Brisbane, QLD 4026, Australia
{son.tran,qing.zhang,anthony.nguyen}@csiro.au
[2] Department of Computing Science, Umeå University, Umeå, Sweden
sonvx@cs.umu.se
[3] Department of Computer Science, FPT University, Hanoi, Vietnam
sonnt69@fe.edu.vn

Abstract. Recurrent neural networks (RNNs) is a useful tool for sequence labelling tasks in natural language processing. Although in practice RNNs suffer a problem of vanishing/exploding gradient, their compactness still offers efficiency and make them less prone to overfitting. In this paper we show that by propagating the prediction of previous labels we can improve the performance of RNNs while keeping the number of parameters in RNNs unchanged and adding only one more step for inference. As a result, the models are still more compact and efficient than other models with complex memory gates. In the experiment, we evaluate the idea on optical character recognition and Chunking which achieve promising results.

Keywords: Natural language processing · Recurrent neural networks
Sequence labelling

1 Introduction

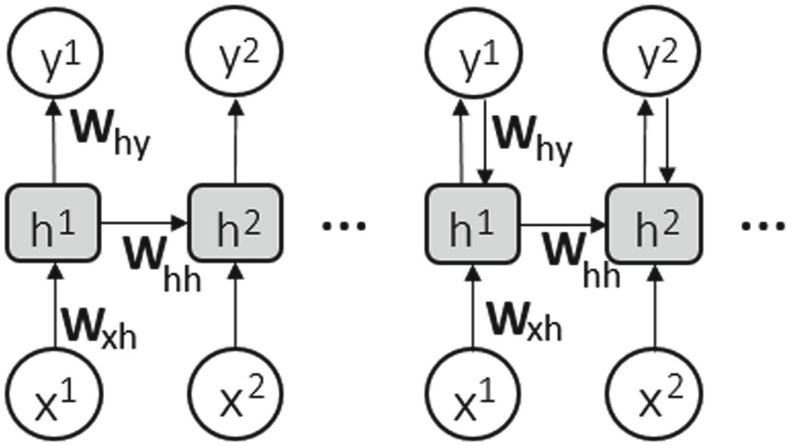
Sequence labelling is a machine learning method which has been widely used for natural language processing tasks. In early work, these tasks attract the use of dynamic Bayesian models such as Hidden Markov Models (HMMs) [21] and Conditional Random Fields (CRFs) [17]. An advantage of such models is the ability to learn relationships between sequence labels, which is useful for temporal reasoning. Recently recurrent neural networks (RNNs) have become a central tool for sequence labelling. An RNN is constructed by rolling an artificial neural network with one hidden layer over time. The hidden layer is connected to itself through recurrent weights. A main advantage of RNNs is the ability to learn temporal representations from data using recurrent hidden layers. Different from dynamic Bayesian models, RNNs assume that the class labels in a sequence are

L. Cheng et al. (Eds.): ICONIP 2018, LNCS 11301, pp. 452–462, 2018.
https://doi.org/10.1007/978-3-030-04167-0_41

independent, given the sequence inputs. This makes inference easier, but with the sacrifice of valuable information: the temporal dependencies of sequence labels.

However, as a type of deep architecture, RNNs suffer the problem of vanishing/exploding gradient which necessitates the use of complex memory gates such as Long-short term memory (LSTM) [14] and Gated recurrent unit (GRU) [3]. However, in many cases, especially when efficiency and compactness are of vital importance, RNNs are more desirable. For example, one would choose an RNN over LSTM for memory-limited devices such as mobile phones and smart sensors if its performance is acceptable. Also, the complexity of LSTM and its variants make it prone to overfitting when training data is small. This situation is very common with natural language processing because of the difficulty in labelling ground truth for large number of sequences. Therefore, it would be useful for an improved version of RNNs which can perform comparably well in comparison to complex models such as GRU and LSTM while keeping the number of parameters remain small in size.

In this paper we show that by propagating the prediction of previous labels we can improve the performance of RNNs. This means that we can keep the number of parameters in RNNs unchanged while adding only one more step for inference. Therefore, the models are still more compact and efficient than other models with complex memory gates.



(a) RNNs (b) Predictive propagation RNNs

Fig. 1. RNNs and ppRNNs

In the experiments, we evaluate RNNs with predictive propagation (ppRNNs) on two sequence labelling tasks: OCR and Chunking. The results show that in most cases predictive propagation help improving the performance of RNNs. We also evaluate the effect of the data sizes on ppRNNs using a POS tagging

dataset. We find that ppRNNs achieve higher accuracy than RNNs, GRUs and LSTMs when small training samples are used. Finally, a synthetic dataset is used to compare the computational time needed for training and inference with ppRNNs to that of GRU and LSTM. It shows that due to the additional step for propagating predictions ppRNNs are more computationally expensive than GRU and LSTMS when the dimension of the output layer is very large, e.g. 400 in the experiment. Fortunately, in practice there are many applications that do not require such high number of classes.

The remainder of the paper is organised as follows. In the next section, we review the literature related to this work. Section 2 introduces the idea of predictive propagation in recurrent neural networks. In Sect. 3, we perform the empirical evaluations. Finally, Sect. 4 concludes the paper and discusses the future extensions.

2 Recurrent Neural Networks with Predictive Propagation

2.1 Graphical Structure

A recurrent neural network for sequence labelling is illustrated in Fig. 1a where $\mathbf{W}_{xh}$ is the weight matrix of connections between input and hidden layers; $\mathbf{W}_{hy}$ is the weight matrix of connections between hidden and output layers; $\mathbf{W}_{hh}$ is the weight matrix of recurrent connections; $\mathbf{b}$, $\mathbf{c}$ are the biases of output units and hidden units respectively.

The predictive propagation recurrent neural network is similar to a recurrent neural network, except that the connections between hidden layer and output layer are bidirectional, as shown in Fig. 1b. Here, the upward direction is used for prediction while the downward direction is for propagating that prediction to the next time step. In the next sections we will show how inference and learning are carried out in this model.

2.2 Inference

Inference in ppRNNs at each time step involves the computation of two states: the state of the output layer in current time step for prediction, and the state of hidden layer that would be used to propagate the current and previous information, including the prediction, to the next step. The details of those steps are in Algorithm 1. Here, f is an activation function and s is the softmax function $s(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{i'} \exp(x_i)}$.

2.3 Learning

We train the RNNs using each sample at a time. In particular for each training pair $\mathbf{x}^{1:T}, \mathbf{y}^{1:T}$ from the dataset we infer $\tilde{\mathbf{y}}^{1:T}$ using the Algorithm 1 and update

Algorithm 1. Predictive propagation

Data: Input: $\mathbf{x}^{1:T}$
Result: Output: $\mathbf{o}^{1:T}$
for $t = 1 : T$ **do**
  $\tilde{\mathbf{h}}^t = f(\mathbf{x}^{t\top}\mathbf{W}_{xh} + \mathbf{h}^{t-1\top}\mathbf{W}_{hh} + \mathbf{c}^\top)$
  $\tilde{\mathbf{y}}^t = s(\tilde{\mathbf{h}}^{t\top}\mathbf{W}_{hy} + \mathbf{b}^\top)$
  $o^t = \arg\max_k \tilde{y}_k^t$
  $\mathbf{h}^t = f(\mathbf{x}^{t\top}\mathbf{W} + \tilde{\mathbf{y}}^{t\top}\mathbf{W}_{hy}^\top + \mathbf{h}^{t-1\top}\mathbf{W}_{hh} + \mathbf{c}^\top)$
end

the parameters by minimising the cross entropy:

$$\mathcal{C} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{l=1}^{L}[y_l^t \log \tilde{y}_l^t + (1 - y_l^t)\log(1 - \tilde{y}_l^t)] \tag{1}$$

where L is the number of classes. A RNN trained by this method is denoted as ppRNN$_p$. Alternatively, with the availability of the true labels, we can use another method to infer $\tilde{\mathbf{y}}^{1:T}$ by replacing $\tilde{\mathbf{y}}^t$ in the last expression in Algorithm 1 with $\mathbf{y}^t$. The state of hidden layer at time t then becomes $\mathbf{h}^t = g(\mathbf{x}^{t\top}\mathbf{W} + \mathbf{y}^{t\top}\mathbf{W}_{hy}{}^\top + \mathbf{h}^{t-1\top}\mathbf{W}_{hh} + \mathbf{c}^\top)$. We denote a ppRNN using this type of inference for learning as ppRNN$_g$.

3 Experiments

3.1 OCR

The MIT OCR dataset[1] is a widely used benchmark for evaluating sequence labelling algorithms [25]. We use two popular partitions from [19] and [1,9]. In the former, called here "ms" for model selection, the data is partitioned into 10 groups, each consisting of a training, validation, and test set. We select models based on performance on the validation sets and report their average accuracy on the test sets. In the latter, here called "cv" for cross-validation, the data is divided into 10 folds in the usual way but without model selection. Each fold in the "cv" partition has ∼6000 training samples, about 10 times larger than each fold in "ms".

Effect of Learning Methods. First, we use the "ms" partition which consists of ten folds, each has ∼600, ∼100, and ∼5400 samples for training, validation, and testing respectively. With such a small number of samples for training we can anticipate the issue of overfitting for large models. Besides, selection of learning methods is also very important. Therefore, we start with testing RNNs with

[1] http://www.seas.upenn.edu/~taskar/ocr/.

different number of hidden units: $hidNum = \{100, 500, 1000, 2000, 5000, 10000\}$; and learning methods: $lrn = \{sgd$: vanilla stochastic gradient descent , *adagrad* [10], *rmsprop*, *adam* [16]$\}$. For each pair of (*hidNum*,*lrn*), we select the best learning rate and activation function based on the performance of validation sets. For early stopping, if performance on a validation set does not improve for 20 epochs then we stop the learning and use the best trained model for testing. As we can see from Fig. 2 the performances of RNNs on the test set drop significantly when the number of hidden units is too high. This makes sense because such overfitting phenomenon is more likely to happen when the models become more complex, i.e. have large number parameters. Overall, adaptive learning method *adam*, which is recently introduced, is shown better than the other methods tested here. *adam* also offers another advantage is that it is fast and works very well with sparse data. Different initial learning rates can be used for *adam* to produce good results but we found that normally 0.001 is good for all cases. In terms of activation function, surprisingly it seems that using *sigmoid* for RNNs performs better than *tanh* despite a study in deep feed-forward neural networks suggesting that the former may suffer from a gradient saturation issue [12].

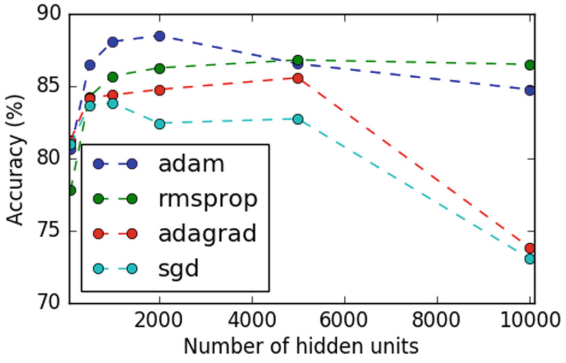


Fig. 2. Performance of RNNs on OCR test set with different learning algorithms and number of hidden units

Overall Results. We compare the performance of SCRBM on the above sequence labelling task with the models: Multiclass support vector machines (SVM-*multiclass*) [6], Structured support vector machines (SVM-*struct*) [26], Max-margin Markov network (M3N) [25], Averaged Perceptron [4], Search-based structured prediction model, a.k.a SEARN [7], Conditional random field (CRF) [17,20], Hidden Markov model (HMM) [21], LogitBoost [11], TreeCRF [8], RTD-RBM [2] (modified to use the inference algorithm proposed in this paper so it works with a sequence labelling task), and state-of-the-art models:

Table 1. Average test set accuracy (%) of various models on the OCR sequence labelling task; "ms" dataset uses model selection and "cv" uses cross-validation without model selection.

Model	ms	cv
ppRNN_g	88.54	95.20
ppRNN_p	**88.60**	95.54
$\text{RNN}_{sigmoid+adam}$	88.44	**95.64**
LSTM	88.28	95.24
GRU	85.77	93.62
RTDRBM	84.54	-
Neural CRF^{CML}	-	95.56
Neural CRF^{LM}	-	95.44
SLE	79.42	-
GBCRF	-	95.36
TreeCRF	-	93.01
LogitBoost	-	90.33
$\text{RNN}_{tanh+sgd}$	77.08	86.67
M3N	74.92	86.54
Perceptron	73.60	-
SEARN	72.98	-
$\text{SVM}_{multiclass}$	71.46	-
SVM_{struct}	78.84	-
HMM	76.30	-
CRF	67.70	85.80

- Structured learning ensemble (SLE) [19]: An optimised ensemble of 7 effective models: SVM-*multiclass*, SVM-*struct*, M3N, Perceptron, SEARN, CRF and HMM.
- Neural CRF [9]: A combination of CRF and deep networks.
- Gradient boosting CRF (GBCRF) [1]: CRF trained by a novel gradient boosting algorithm.

For the "ms" partition, to determine the best model for the task, a grid search was carried out where the number of hidden units, learning rate, activation function are selected similarly as in Sect. 3.1. For the "cv" partition, since model selection is not possible, we use 2000 hidden units and set the initial learning rate for *adam* [16] as 0.001. Each model is trained in 30 epochs.

Table 1 summarises the results and shows that with a right choice of activation function (*sigmoid*) and learning method (*adam*) RNNs can perform very well in this dataset. Although improvement has not been seen in "cv" partition ppRNNs achieve higher accuracy than other methods in "ms" partition where

the number of training samples is smaller. The results also imply that propagating prediction probabilities (ppRNN$_p$) seems slightly better than propagating ground truth (ppRNN$_g$), during the learning (Table 2).

Table 2. Features extraction for Conll2000 Chunking task (see [23])

$w_{t-\delta} = w$
w_t matches [A-Z][a-z]+
w_t matches [A-Z]
w_t matches [A-Z]+
w_t matches [A-Z]+[a-z]+[A-Z]+[a-z]
w_t matches .*[0-9].*
w_t appears in list of first names, last names, company names, days, months, or geographic entities
w_t is contained in a lexicon of words with POS T (from Brill tagger)
$T_t = T$
$q_k(x, t + \delta)$ for all k and $\delta \in [-3, 3]$

3.2 CoNLL 2000 Chunking

The CoNLL 2000 shared task is a benchmark data for sequence labelling with a focus on Chunking. The dataset consists of 8,936 and 2012 sentences for training and testing respectively. For this data we use the binary features from [23].

In this experiment, we use 50000 most common features from the training set. The motivation behind the selection of this type of features over word2vec features is to exploit the efficiency of RNNs. Although word2vec features are smaller in terms of size, generic approaches such as RNN, GRU, LSTM do not perform well, and therefore more complex variants, i.e. biLSTM [13], bi-LSTM-CRF [15] and CNN-biLSTM-CRF [18] are needed when using word2vec features as their sole input. Here, the compactness and efficiency of RNNs make it easier to work with such highly dimensional hand-crafted features, which in this case perform better than word2vec features .

Since the dataset only includes training and testing samples we do not perform model selection and early stopping. Instead, we set the number of hidden units to be 500 and the number of training epochs to be 50. Such hyper-parameters are chosen based on the capacity of our computers used in this experiment. We also use *adam* to take the advantage of sparsity of the features, the initial learning rate for it is 0.001.

As we can see from Table 3, the features are so effective that even a simple RNN can give a better result than many other approaches using different

Table 3. F1 score in CoNLL 2000 chunking data.

Model	F1 score
ppRNN$_g$	**95.319**
ppRNN$_p$	**95.302**
LSTM	95.118
GRU	94.719
RNN	95.085
Suzuki et. al. [24]	95.15
Huan et. al. [15]	94.46
Sun et. al. [22]	94.34
Collobert et. al. [5]	94.32
Tsuruoka et. al. [27]	93.81

features. Again, we show that predictive propagation can help to improve the performance. It is also worth noting that the ppRNNs in this case only needs ∼4 h to train while the *GRU* and *LSTM* take more than ∼10 h, using NVIDIA Quadro P1000. Although the other models we compare to in Table 3 are applicable to the hand-crafted features above, in this experiment we show that RNNs can perform very well while being very efficient.

Table 4. Test accuracy of ppRNNs, RNN, GRU and LSTM on POS dataset with different number of training samples.

Model	500	1000	2000	4000	8000
ppRNN$_g$	**88.703**	**90.847**	91.757	92.122	93.336
ppRNN$_p$	88.646	90.767	91.781	92.244	**93.419**
RNN	88.293	90.755	**91.808**	92.303	93.380
LSTM	88.529	90.585	91.606	**92.445**	**93.414**
GRU	88.571	90.526	91.169	92.276	93.046

3.3 POS Tagging: Effect of Training Size

In this experiment we use a POS tagging dataset with different training sizes of 500, 1000, 4000, 8000 samples to compare the effectiveness of ppRNNs with RNN, GRU, and LSTM. The samples are obtained from Penn Treebank 2002 dataset. Model selection and early stopping are done by leaving out 10% of the training set for validation. The test set consists of ∼1600 samples and is applied to all cases.

Similar to the Chunking task above, we also use binary features for each token in POS task. The results in Table 4 show that ppRNNs perform better than the others when the training data is small.

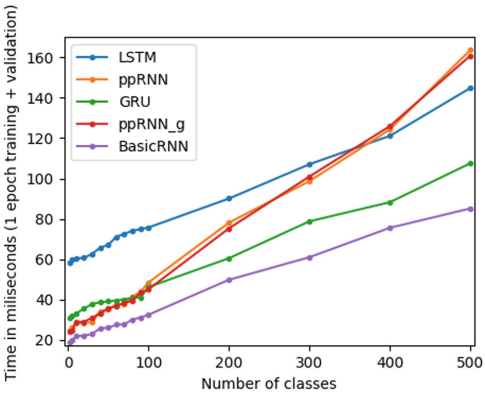


Fig. 3. Computational time.

3.4 Computational Time

Finally, in order to compare the efficiency of ppRNNs over other types of RNNs we use a synthetic dataset of 1000 training samples and 1000 validation samples. Each sample has the length of 100 and the input's dimension is 1000. We train each model in one epoch and evaluate it on the validation set. All RNNs have 100 hidden units. Since the computation in ppRNNs requires the propagation of the prediction back to the hidden layer we test the computational time of the models on the data with different number of classes using a PC with 4 Intel Core^{TM} i5-6500 CPUs @ 3.20 Hz and 32 GiB RAM. From Fig. 3 we can see that ppRNN$_g$ and ppRNN$_p$ takes similar time to learn and infer, and also that when the number of labels grows larger, i.e. ∼400, ppRNNs become slower than LSTM. However, it is not common to see such a large number of labels in sequence labelling in practice.

4 Conclusion

We have shown an empirical study on propagating prediction of output layer in RNNs for sequence labelling. In most cases, it helps improve the performance of RNNs while maintaining the compactness. The computational time of the proposed models is practically efficient in comparison with other RNNs having complex memory gates.

References

1. Chen, T., Singh, S., Taskar, B., Guestrin, C.: Efficient second-order gradient boosting for conditional random fields. In: 18th International Conference on Artificial Intelligence and Statistics, vol. 38, pp. 147–155. PMLR, San Diego (2015)
2. Cherla, S., Tran, S.N., d'Garcez, A., Weyde, T.: Discriminative learning and inference in the recurrent temporal RBM for melody modelling. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2015)
3. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Conference on Empirical Methods in Natural Language Processing, pp. 1724–1734 (2014)
4. Collins, M.: Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In: ACL-2002 Conference on Empirical Methods in Natural Language Processing, vol. 10, pp. 1–8. Association for Computational Linguistics, Stroudsburg (2002)
5. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12**, 2493–2537 (2011)
6. Crammer, K., Singer, Y.: On the algorithmic implementation of multiclass kernel-based vector machines. J. Mach. Learn. Res. **2**, 265–292 (2002)
7. Daumé III, H., Langford, J., Marcu, D.: Search-based structured prediction. Mach. Learn. **75**(3), 297–325 (2009)
8. Dietterich, T.G., Hao, G., Ashenfelter, A.: Gradient tree boosting for training conditional random fields. J. Mach. Learn. Res. **9**(2), 2113–2139 (2008)
9. Do, T., Artieres, T.: Neural conditional random fields. In: 13th International Conference on Artificial Intelligence and Statistics, vol. 9, pp. 177–184. PMLR, Sardinia (2010)
10. Duchi, J., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. **12**, 2121–2159 (2011)
11. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Ann. Stat. **28**, 337–407 (2000)
12. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: 13th International Conference on Artificial Intelligence and Statistics, vol. 9, pp. 249–256. PMLR, Sardinia (2010)
13. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM networks. In: 2005 IEEE International Joint Conference on Neural Networks, Montreal, Quebec, Canada, vol. 4, pp. 2047–2052 (2005)
14. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
15. Huang, Z., Xu, W., Yu, K.: Bidirectional LSTM-CRF models for sequence tagging. CoRR abs/1508.01991 (2015)
16. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. CoRR abs/1412.6980 (2014)
17. Lafferty, J.D., McCallum, A., Pereira, F.C.N.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: 18th International Conference on Machine Learning, pp. 282–289. Morgan Kaufmann Publishers Inc., San Francisco (2001)
18. Ma, X., Hovy, E.: End-to-end sequence labeling via bi-directional LSTM-CNNS-CRF. In: 54th Annual Meeting of the Association for Computational Linguistics, pp. 1064–1074. Association for Computational Linguistics (2016)

19. Nguyen, N., Guo, Y.: Comparisons of sequence labeling algorithms and extensions. In: 24th International Conference on Machine Learning, pp. 681–688. ACM, New York (2007)
20. Peng, F., McCallum, A.: Information extraction from research papers using conditional random fields. Inf. Process. Manag. **42**(4), 963–979 (2006)
21. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. In: Readings in Speech Recognition, pp. 267–296. Elsevier, San Francisco (1990)
22. Sun, X., Morency, L.P., Okanohara, D., Tsujii, J.: Modeling latent-dynamic in shallow parsing: a latent conditional model with improved inference. In: 22nd International Conference on Computational Linguistics, pp. 841–848. Association for Computational Linguistics, Stroudsburg (2008)
23. Sutton, C., McCallum, A., Rohanimanesh, K.: Dynamic conditional random fields: factorized probabilistic models for labeling and segmenting sequence data. J. Mach. Learn. Res. **8**, 693–723 (2007)
24. Suzuki, J., Isozaki, H.: Semi-supervised sequential labeling and segmentation using giga-word scale unlabeled data. In: ACL-2008: HLT, pp. 665–673. The Association for Computer Linguistics (2008)
25. Taskar, B., Guestrin, C., Koller, D.: Max-margin Markov networks. In: Advances in Neural Information Processing Systems, vol. 16, p. 25 (2004)
26. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. J. Mach. Learn. Res. **6**, 1453–1484 (2005)
27. Tsuruoka, Y., Miyao, Y., Kazama, J.: Learning with lookahead: can history-based models rival globally optimized models? In: 15th Conference on Computational Natural Language Learning, pp. 238–246. Association for Computational Linguistics, Stroudsburg (2011)